بسمه تعالی

فرشید حسین‌زاده ۹۸۲۱۲۰۳ مینی پروژه شماره یک، بخش اول

سوال اول)

بخش ۱.

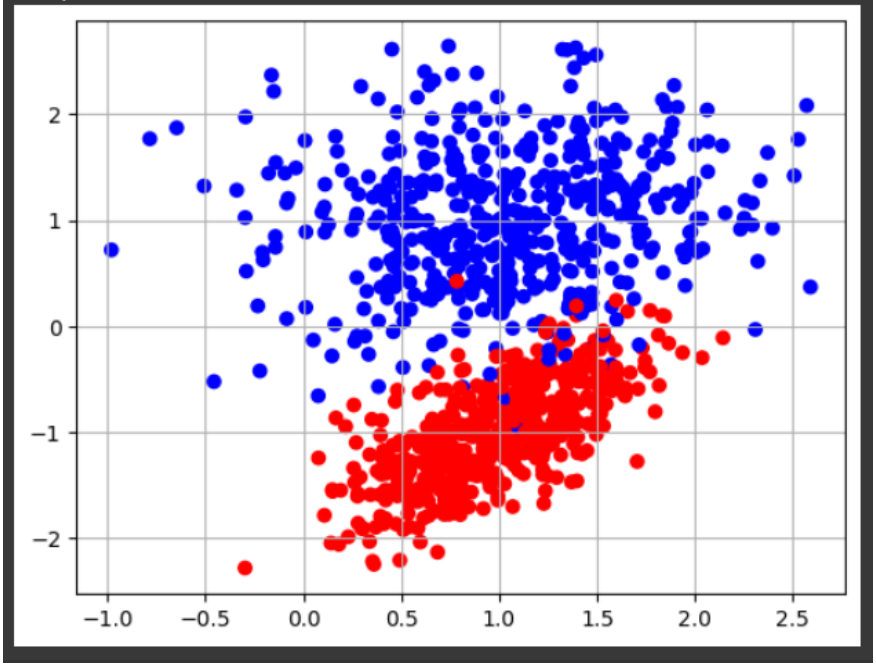
در این بخش با استفاده از sklearn.datasets، یک دیتاست با ۱۰۰۰ نمونه، ۲ کلاس و ۲ ویژگی تولید شده است:

```
[2] import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd
    from sklearn.datasets import make_classification

    X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, n_classes=2, random_state=3)
    print(X.shape, y.shape)

    plt.grid(True)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='bwr')
```

برای این منظور همانطور که در تصویر قابل ملاحظه است، ابتدا کتابخانه‌های موردنیاز را import می‌کنیم، مانند: make_classification و numpy و ... سپس با استفاده از make_classification، دیتاست را تولید می‌کنیم. n_sample بیانگر تعداد نمونه (۱۰۰۰)، n_features بیانگر تعداد ویژگی (۲) و n_classes بیانگر تعداد کلاس‌ها (۲) هستند. از آنجا که رابطه‌ی n_informative + n_redundant + n_repeated < n_features همواره باید برقرار باشد، مقادیر این پارامترها مانند تصویر قراره داده شده‌اند تا از حالت پیش‌فرض خارج شده و خطا دریافت نکنیم. n_cluster_per_class هم برابر یک قرار داده شده تا هر کلاس به یک خوشه و نه بیشتر تبدیل شود. مقدار random_state هم برابر با دو رقم آخر شماره دانشجویی (۰۳) قرار داده شده است.

در شکل زیر خروجی این بخش آورده شده که در ابتدا میتوان ابعاد X و y را دید و سپس نحوه چیدمان و نحوه تولید دیتاست را مشاهده کرد. در این قسمت، cmap='bwr' باعث تغییر رنگ نقطه‌های کلاس‌های مختلف (آبی و قرمز شدن آن‌ها) شده است.

```
(1000, 2) (1000,)
<matplotlib.collections.PathCollection at 0x7b370ede85b0>
```



بخش ۲.

در این بخش با ابتدا با LogisticRegression به عنوان model1 و سپس با استفاده از SGDClassifier به عنوان model2 به تفکیک داده‌ها می‌پردازیم.

```
[5]  from sklearn.linear_model import LogisticRegression, SGDClassifier
     from sklearn.model_selection import train_test_split
```

در ادامه با استفاده از train_test_split داده‌ها را به نسبت ۸۰ به ۲۰ به ترتیب به train و test تقسیم می‌کنیم و برای اطمینان از صحت انجام این عملیات، از دستور shape. استفاده می‌نماییم:

```
[6]  x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)
     x_train.shape, x_test.shape, y_train.shape, y_test.shape

     ((800, 2), (200, 2), (800,), (200,))
```

سپس model1 را برابر با LogisticRegression قرار می‌دهیم؛ برای solver، با توجه به اطلاعاتی که در سایت scikit-learn نوشته شده بود، saga انتخاب شد زیرا با توجه به اطلاعات سایت، برای دیتاست‌هایی با تعداد سمپل زیاد، سرعت همگرایی زیادی دارد. همچنین مقدار max_iter هم ۲۰ انتخاب شد هرچند مقدار پیش فرض آن ۱۰۰ است ولی با چک کردن خروجی (همگرا شدن آن) و دقت حاصله روی داده‌های train و

test، از مقدار ۲۰ به بعد هم همگرا شده و دقت ثابت می‌شود پس استفاده از مقدار بزرگتری مورد نیاز نیست. (تا مقدار ۲۰۰۰ امتحان شد و نتیجه تغییری نکرد)

در ادامه با متود fit. و قرار دادن داده‌های train در آن، مدل را train می‌کنیم و در ادامه با predict. و قرار دادن تنها x های تست در آن، پیشبینی‌های مدل را ملاحظه می‌کنیم و با مقدار واقعی لیبل (y_test) مقایسه می‌کنیم:

```
[15] model1 = LogisticRegression(solver='saga', max_iter=20, random_state=3)
     model1.fit(x_train, y_train)
     model1.predict(x_test), y_test
```

```
(array([1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
        1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
        0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
        1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0,
        1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
        1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1,
        0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
        1, 0]),
 array([1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
        1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
        0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
        1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0,
        1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
        1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
        1, 0]))
```

پس از آن ابتدا با متود score. دقت مدل برای داده‌های train و پس از آن دقت مدل برای داده‌های test را چک می‌کنیم:

```
[16] model1.score(x_train, y_train)
```

```
0.96875
```

```
[17] model1.score(x_test, y_test)
```

```
0.965
```

در ادامه برای طبقه‌بند دوم (model2) همین مراحل را طی می‌کنیم. برای SGDClassifier، نحوه محاسبه loss را برابر با log_loss قرار می‌دهیم تا از همان روش گفته شده در کلاس استفاده شود، مقدار max_iter از پیش فرض آن تغییر کرده و بر روی ۱۰ قرار داده شده زیرا هم همگرا می‌شود و هم مقدار دقت با افزایش آن نیز ثابت می‌ماند. مقدار learning_rate از پیش فرض آن که طبق گفته سایت، optimal است به یک مقدار

ثابت (۰.۲) که نه بسیار کم است که مدل به کندی کار کند و نه بسیار زیاد که خطر همگرا نشدن مدل را تهدید کند، تغییر پیدا کرده است. نحوه استفاده از fit. و predict. هم مانند قبل است:

```
model2 = SGDClassifier(loss='log_loss', max_iter=10, learning_rate='constant', eta0=0.2, random_state=3)
model2.fit(x_train, y_train)
model2.predict(x_test), y_test
```

```
(array([1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       1, 0]),
 array([1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       1, 0]))
```

بررسی دقت مدل: (مانند قبل)

```
[29] model2.score(x_train, y_train)

     0.975

[30] model2.score(x_test, y_test)

     0.965
```

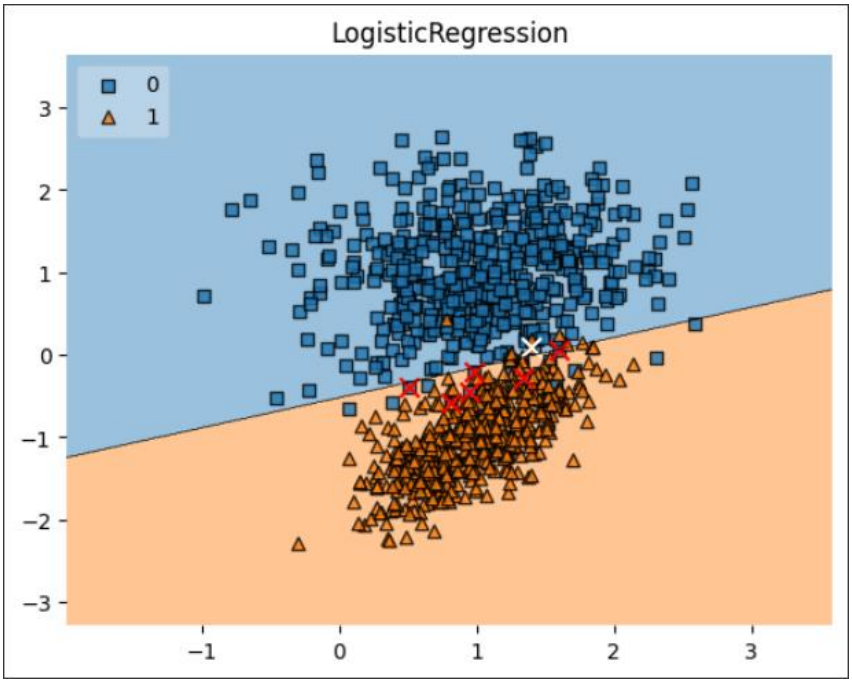
بخش ۳.

در این بخش خط و نواحی تصمیم‌گیری را مشخص می‌کنیم. برای این کار، از plot_decision_regions، به عنوان کتابخانه آماده استفاده می‌کنیم که در داخل خود، پارامترهای X و y را که متعلق به همان دیتاست تولید شده است و مدل مورد استفاده (در ابتدا model1 که با LogisticRegression کار می‌کند) را در پارامتر clf دریافت می‌کند. در ادامه برای مشخص کردن نقاطی که به درستی توسط مدل پیشبینی نمی‌شوند، misclassified را تعریف می‌کنیم که در واقع X را برای نقاطی که مقدار y پیشبینی شده آنها برابر با لیبل اصلی آن‌ها نیست پیدا می‌کند؛ سپس یک نمودار scatter از نقاطی که اشتباه پیشبینی شده‌اند میسازیم به این

شکل که ابتدا نقاطی را که متعلق به کلاس صفر هستند ولی اشتباه تشخیص داده شده‌اند را تعیین کرده و با استفاده از دستور ([:, 0][0 ==) مختصات x آن‌ها و با دستور ([:, 1][0 ==)، مختصات y آن‌ها را بدست می‌آوریم. همین فرایند را برای کلاس یک نیز تکرار می‌کنیم. نحوه نمایش نقاط غلط با یک علامت ضربدر با سایز ۸۰ می‌باشد که نقاط غلط از کلاس صفر با رنگ قرمز و از کلاس یک با رنگ سفید نشان داده می‌شوند. سپس نمودار را عنوان می‌زنیم و نمایش می‌دهیم:

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X, y, clf=model1, legend=2)

misclassified = x_test[y_test != model1.predict(x_test)]

plt.scatter(misclassified[y_test[y_test != model1.predict(x_test)] == 0][:, 0],
            misclassified[y_test[y_test != model1.predict(x_test)] == 0][:, 1],
            marker='x', s=80, c='red')

plt.scatter(misclassified[y_test[y_test != model1.predict(x_test)] == 1][:, 0],
            misclassified[y_test[y_test != model1.predict(x_test)] == 1][:, 1],
            marker='x', s=80, c='white')

plt.title('LogisticRegression')
plt.show()
```

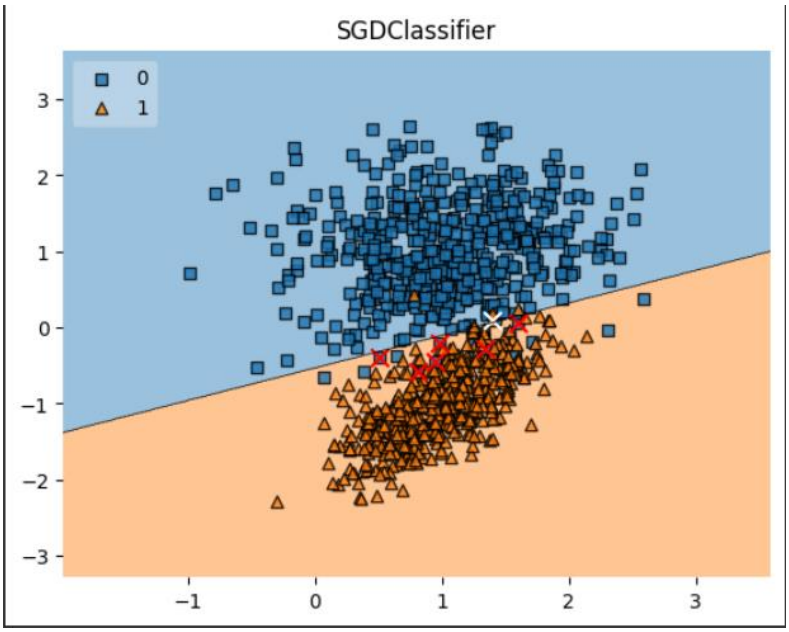
نمودار خروجی:



همین فرایند را برای model2 (SGDClassifier) نیز تکرار می‌کنیم:

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X, y, clf=model2, legend=2)

misclassified = x_test[y_test != model1.predict(x_test)]

plt.scatter(misclassified[y_test[y_test != model1.predict(x_test)] == 0][:, 0],
            misclassified[y_test[y_test != model1.predict(x_test)] == 0][:, 1],
            marker='x', s=80, c='red')

plt.scatter(misclassified[y_test[y_test != model1.predict(x_test)] == 1][:, 0],
            misclassified[y_test[y_test != model1.predict(x_test)] == 1][:, 1],
            marker='x', s=80, c='white')

plt.title('SGDClassifier')
plt.show()
```

خروجی:



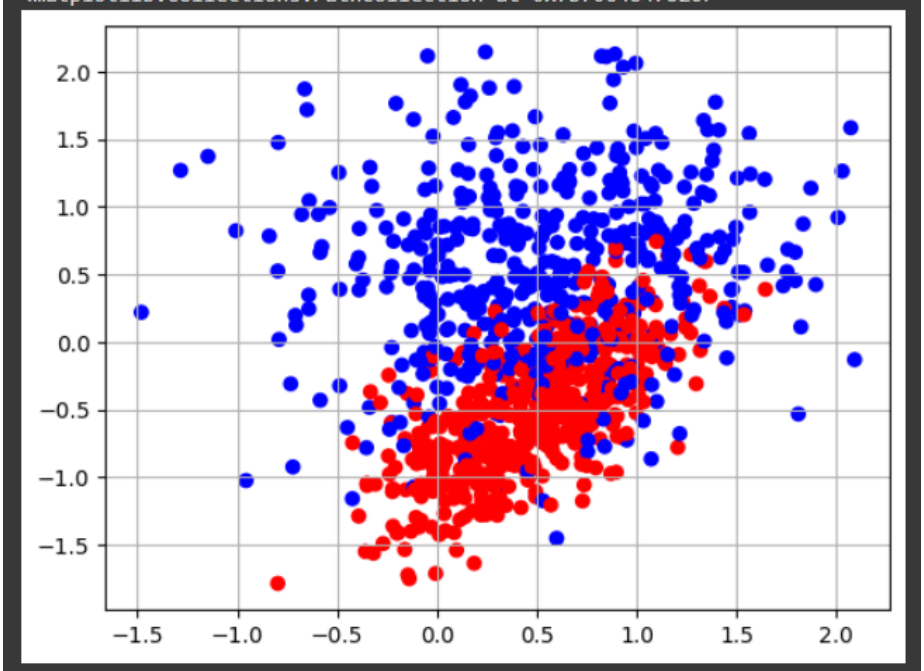
بخش ۴.

برای چالش‌برانگیزتر کردن دیتاست، می‌توان کاری کرد که داده‌های دو کلاس بیشتر درهم‌آمیخته باشند یعنی class_sep که پیش فرض ۱ است را کم‌تر کنیم. مثلا در ادامه از مقدار ۰.۵ استفاده می‌کنیم:

```
[62] X_harder, y_harder = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, n_classes=2, random_state=3, class_sep=0.5)
     print(X_harder.shape, y_harder.shape)

     plt.grid(True)
     plt.scatter(X_harder[:, 0], X_harder[:, 1], c=y, cmap='bwr')
```

که داده‌ها به شکل زیر درمی‌آیند:

```
(1000, 2) (1000,)
<matplotlib.collections.PathCollection at 0x7b7064847820>
```



حال همه‌ی مراحل قبل را تکرار می‌کنیم:

```
[63] x_harder_train, x_harder_test, y_harder_train, y_harder_test = train_test_split(X_harder, y_harder, test_size=0.2, random_state=3)
     x_harder_train.shape, x_harder_test.shape, y_harder_train.shape, y_harder_test.shape

     ((800, 2), (200, 2), (800,), (200,))

[70] model1_harder = LogisticRegression(solver='saga', max_iter=200, random_state=3)
     model1_harder.fit(x_harder_train, y_harder_train)
     model1_harder.predict(x_harder_test), y_harder_test

     (array([0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0,
             1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
             1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1,
             1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
             0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
             1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1,
             1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0,
             1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
             0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0,
             1, 0]),
      array([1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
             1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1,
             1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
             1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
             0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
             1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0,
             1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
             1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1,
             1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
             1, 0]))
```

فقط تنها تکرار نسبت به حالت قبل بیشتر کردن مراحل تکرار است که البته با ۲۰ بار طبق حالت قبل نیز نتیجه یکسانی مشاهده می‌شود.

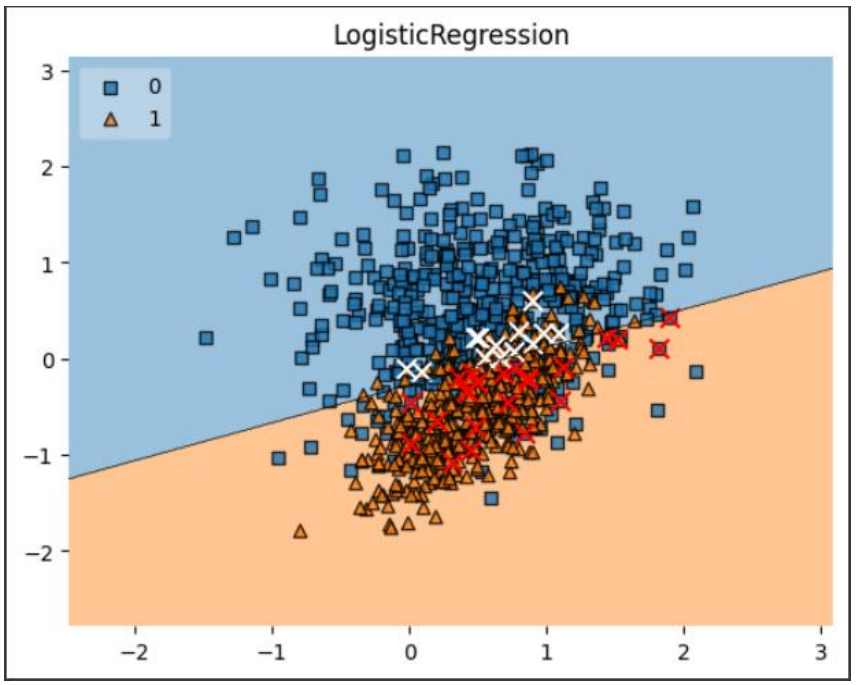
دقت مدل برای داده‌های train و test:

```
[71] model1_harder.score(x_harder_train, y_harder_train)
     0.825

[73] model1_harder.score(x_harder_test, y_harder_test)
     0.82
```

مثل قبل نواحی تصمیم‌گیری را نیز رسم می‌کنیم:

```
plot_decision_regions(X_harder, y_harder, clf=model1_harder, legend=2)

misclassified = x_harder_test[y_harder_test != model1_harder.predict(x_harder_test)]

plt.scatter(misclassified[y_harder_test[y_harder_test != model1_harder.predict(x_harder_test)] == 0][:, 0],
            misclassified[y_harder_test[y_harder_test != model1_harder.predict(x_harder_test)] == 0][:, 1],
            marker='x', s=80, c='red')

plt.scatter(misclassified[y_harder_test[y_harder_test != model1_harder.predict(x_harder_test)] == 1][:, 0],
            misclassified[y_harder_test[y_harder_test != model1_harder.predict(x_harder_test)] == 1][:, 1],
            marker='x', s=80, c='white')

plt.title('LogisticRegression')
plt.show()
```

خروجی:

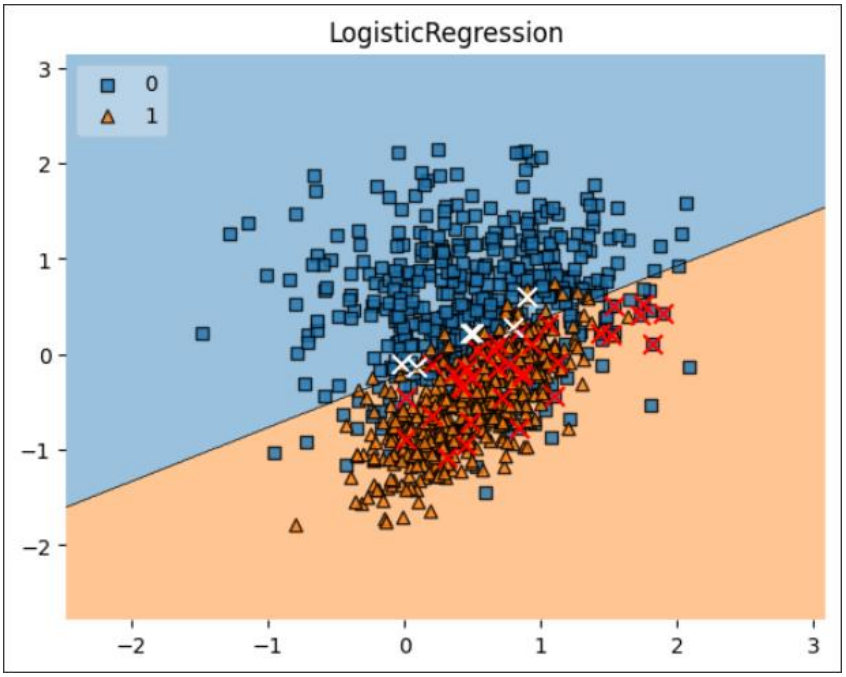


همه‌ی این مراحل را برای SGDClassifier نیز تکرار می‌کنیم که برای جلوگیری از شلوغی بیش از حد در ادامه گزارشکار آورده نشده و صرفا دقت مدل و نتیجه ناحیه تصمیم‌گیری آورده شده و تمام مراحل در فایل گوگل کولب قرار دارد. البته در این قسمت با تعداد تکرار ۱۰ همگرایی و دقت را از دست می‌دادیم پس تعداد تکرار بر روی ۱۰۰ قرار داده شده است.

دقت‌ها:

```
[81] model2_harder.score(x_harder_train, y_harder_train)
     0.81375
[82] model2_harder.score(x_harder_test, y_harder_test)
     0.805
```

نواحی تصمیم‌گیری:



بخش ۵.

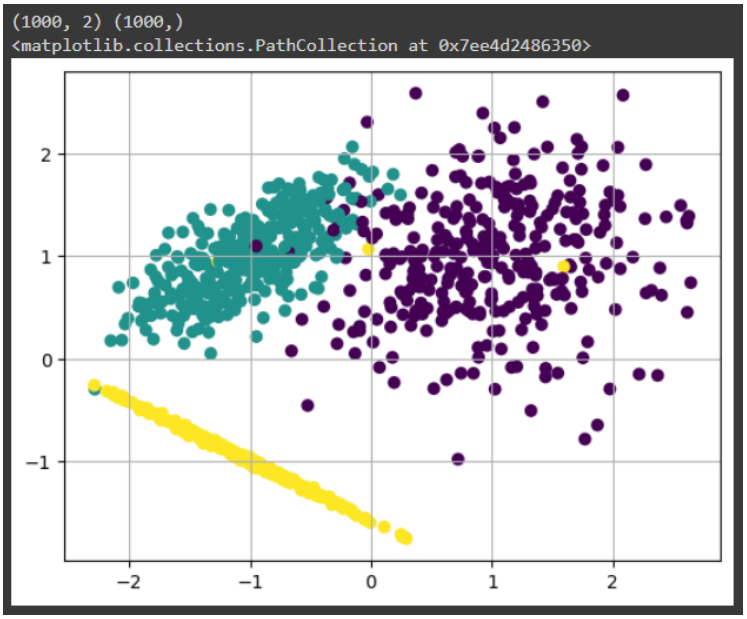
با اضافه کردن یک کلاس به کلاس‌های قسمت اول، همچنان می‌توان با استفاده از کتابخانه‌های آماده به طبقه بندی داده‌ها پرداخت. مراحل پیاده‌سازی این قسمت در ادامه آورده شده است:

تولید دیتاست:

```
X1, y1 = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, class_sep=1, n_clusters_per_class=1, n_classes=3, random_state=3)
print(X1.shape, y1.shape)

plt.grid(True)
plt.scatter(X1[:, 0], X1[:, 1], c=y)
```

دیتاست:

```
(1000, 2) (1000,)
<matplotlib.collections.PathCollection at 0x7ee4d2486350>
```



تقسیم بندی داده‌ها به train و test و ساخت مدل با استفاده از LogisticRegression:

```
x1_train, x1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2, random_state=3)
x1_train.shape, x1_test.shape, y1_train.shape, y1_test.shape
```

```
((800, 2), (200, 2), (800,), (200,))
```

```
model1_5 = LogisticRegression(solver='saga', max_iter=20, random_state=3)
model1_5.fit(x1_train, y1_train)
model1_5.predict(x1_test), y1_test
```

```
(array([0, 2, 0, 2, 0, 2, 0, 1, 0, 1, 2, 0, 1, 0, 2, 2, 1, 1, 2, 1, 0, 2,
        1, 1, 2, 2, 1, 2, 1, 0, 0, 1, 2, 2, 2, 0, 0, 1, 0, 2, 2, 0, 2, 1,
        1, 0, 1, 2, 1, 1, 1, 2, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 2, 2, 2,
        1, 1, 2, 2, 1, 2, 0, 0, 1, 1, 1, 0, 2, 2, 1, 2, 1, 2, 0, 1, 0, 0,
        1, 0, 1, 1, 0, 2, 0, 2, 1, 1, 2, 0, 0, 2, 0, 1, 0, 0, 2, 1, 2, 0,
        1, 2, 2, 1, 0, 1, 1, 0, 1, 1, 2, 2, 0, 1, 1, 0, 1, 2, 1, 1, 1, 1,
        0, 0, 2, 0, 1, 2, 0, 2, 2, 1, 0, 1, 2, 2, 1, 2, 0, 2, 1, 0, 2, 0,
        0, 1, 2, 2, 2, 1, 1, 0, 0, 0, 0, 0, 1, 1, 2, 1, 0, 2, 1, 1, 0, 1,
        2, 0, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 1, 1, 0, 2, 0, 0, 1, 1, 0,
        2, 1]),
 array([0, 2, 0, 2, 0, 2, 0, 0, 0, 1, 2, 0, 1, 2, 2, 2, 1, 1, 2, 1, 0, 2,
        1, 1, 2, 2, 1, 2, 1, 0, 0, 0, 2, 2, 2, 0, 0, 1, 0, 2, 2, 0, 2, 1,
        1, 0, 1, 2, 1, 1, 1, 2, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 2, 2, 2,
        1, 1, 2, 2, 1, 2, 0, 0, 1, 1, 1, 0, 2, 2, 1, 2, 1, 2, 0, 1, 0, 0,
        1, 0, 1, 1, 0, 2, 0, 2, 1, 1, 2, 0, 0, 2, 0, 1, 0, 0, 2, 1, 2, 0,
        1, 2, 2, 1, 0, 1, 1, 0, 1, 1, 2, 2, 0, 1, 1, 0, 1, 2, 1, 1, 1, 1,
        0, 0, 2, 0, 1, 2, 0, 2, 2, 1, 0, 1, 2, 2, 1, 2, 0, 2, 1, 0, 2, 0,
        0, 1, 2, 2, 2, 1, 1, 0, 0, 0, 0, 0, 1, 1, 2, 1, 0, 2, 1, 1, 0, 1,
        2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 1, 1, 0, 2, 0, 0, 1, 0, 0,
        2, 1]))
```

دقت مدل برای داده‌های train و test:

```
model1_5.score(x1_train, y1_train)
```

```
0.98125
```

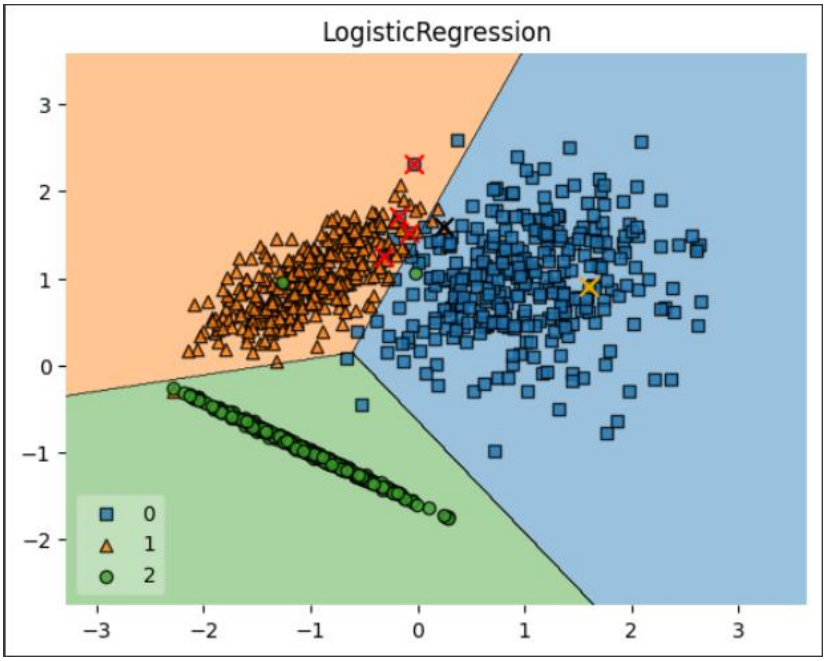
```
model1_5.score(x1_test, y1_test)
```

```
0.97
```

جداسازی نواحی طبقه‌بندی:

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X1, y1, clf=model1_5, legend=3)

misclassified = x1_test[y1_test != model1_5.predict(x1_test)]

plt.scatter(misclassified[y1_test[y1_test != model1_5.predict(x1_test)] == 0][:, 0],
            misclassified[y1_test[y1_test != model1_5.predict(x1_test)] == 0][:, 1],
            marker='x', s=80, c='red')

plt.scatter(misclassified[y1_test[y1_test != model1_5.predict(x1_test)] == 1][:, 0],
            misclassified[y1_test[y1_test != model1_5.predict(x1_test)] == 1][:, 1],
            marker='x', s=80, c='black')

plt.scatter(misclassified[y1_test[y1_test != model1_5.predict(x1_test)] == 2][:, 0],
            misclassified[y1_test[y1_test != model1_5.predict(x1_test)] == 2][:, 1],
            marker='x', s=80, c='orange')

plt.title('LogisticRegression')
plt.show()
```

نتیجه جداسازی:



لازم به ذکر است چون منطق کدزنی در همه‌ی مراحل این قسمت مانند قبل بود، توضیح اضافی برای این مرحله داده نشده و صرفا به نمایش کدها و نتایج آن‌ها بسنده شده است.

سوال دوم)

بخش ۱.

این دیتاست برگرفته از تصاویری است که با یک روش احراز هویت برای اسکناس‌ها بدست آمده‌اند. این دیتاست، یک دیتاست چند متغیره است که از داده‌های واقعی گرفته شده که به منظور طبقه‌بندی مورد استفاده قرار می‌گیرد و دارای ۱۳۷۲ نمونه است.

بارگذاری در گوگل کولب:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1u4DSb_5xPFW9X_5i3SRq8hpBlQ1I5K0j
```

سپس کتابخانه‌های مورد نیاز را اضافه می‌کنیم و در ادامه به دلیل اینکه در این دیتاست، feature ها و لیبل از هم جدا نشده‌اند و همه‌ی آنها به صورت یک فایل txt. و پشت سر هم قرار دارند، نیاز داریم که به نحوی آن‌ها را از هم متمایز کنیم که با دستور زیر هم این کار را انجام می‌دهیم و هم dataframe را ایجاد می‌کنیم:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
df = pd.read_csv('data_banknote_authentication.txt', names=['Feature1', 'Feature2', 'Feature3', 'Feature4', 'Class'])
df
```

	Feature1	Feature2	Feature3	Feature4	Class
0	3.62160	8.66610	-2.8073	-0.44699	0
1	4.54590	8.16740	-2.4586	-1.46210	0
2	3.86600	-2.63830	1.9242	0.10645	0
3	3.45660	9.52280	-4.0112	-3.59440	0
4	0.32924	-4.45520	4.5718	-0.98880	0
...	...	...	...	...	...
1367	0.40614	1.34920	-1.4501	-0.55949	1
1368	-1.38870	-4.87730	6.4774	0.34179	1
1369	-3.75030	-13.45860	17.5932	-2.77710	1
1370	-3.56370	-8.38270	12.3930	-1.28230	1
1371	-2.54190	-0.65804	2.6842	1.19520	1

1372 rows × 5 columns

بخش ۲.

بر زدن داده‌ها قبل از تقسیم آن‌ها به train و test به دلایل زیادی دارای اهمیت است مثلا اگر داده‌ها به روش خاصی مرتب شده باشند (مثل داده‌های ما که در ابتدا کلاس صفر چیده شده و سپس کلاس یک) و ما بدون بر زدن آن‌ها را تقسیم کنیم، ممکن است مدل با توجه به توالی داده‌ها آموزش ببیند و نتواند به خوبی و به بهترین شکل عمل کند. همچنین بر زدن می‌تواند از وابسته شدن مدل به ترتیب داده‌ها جلوگیری کند و باعث جلوگیری از Overfitting نیز می‌شود.

برای بر زدن داده‌ها از دستور زیر استفاده می‌کنیم که در آن frac=1 به این معناست که همه‌ی داده‌ها در فرایند بر زدن شرکت کنند و برای مثال اگر ما frac=0.5 قرار می‌دادیم، پنجاه درصد داده‌ها در فرایند بر زدن شرکت می‌کردند. نتیجه فرایند بر زدن نیز در عکس زیر قابل مشاهده می‌باشد:

```
df_shuffled = df.sample(frac=1, random_state=3)
df_shuffled
```

	Feature1	Feature2	Feature3	Feature4	Class
1258	-0.62043	0.55870	-0.38587	-0.66423	1
712	4.79650	6.98590	-1.99670	-0.35001	0
750	4.04220	-4.39100	4.74660	1.13700	0
1295	-4.94470	3.30050	1.06300	-1.44400	1
888	-2.57010	-6.84520	8.99990	2.13530	1
...	...	...	...	...	...
789	1.05520	1.18570	-2.64110	0.11033	1
256	3.09340	-2.91770	2.22320	0.22283	0
968	-1.95550	0.20692	1.24730	-0.37070	1
952	-1.28460	3.27150	-1.76710	-3.26080	1
1273	-2.36750	-0.43663	1.69200	-0.43018	1

1372 rows × 5 columns

در ادامه داده‌های مربوط به feature را به متغیر X و داده‌های Class را به متغیر y نسبت می‌دهیم:

```
X = df_shuffled[['Feature1', 'Feature2', 'Feature3', 'Feature4']].values
y = df_shuffled['Class'].values
X, y
```

```
(array([[-0.62043,  0.5587 , -0.38587, -0.66423],
        [ 4.7965 ,  6.9859 , -1.9967 , -0.35001],
        [ 4.0422 , -4.391  ,  4.7466 ,  1.137  ],
        ...,
        [-1.9555 ,  0.20692,  1.2473 , -0.3707 ],
        [-1.2846 ,  3.2715 , -1.7671 , -3.2608 ],
        [-2.3675 , -0.43663,  1.692  , -0.43018]]),
 array([1, 0, 0, ..., 1, 1, 1]))
```

در ادامه داده ها را به دو بخش train و test با نسبت ۸۰ به ۲۰ تقسیم می‌کنیم:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((1097, 4), (275, 4), (1097,), (275,))
```

بخش ۳.

در ابتدا توابع مورد نیاز برای طبقه‌بندی بدون استفاده از کتابخانه‌های آماده را تعریف می‌کنیم. (اکثر توابع مانند ویدیو آموزشی قرار داده شده، نوشته شده‌اند پس از آوردن توضیحات اضافه برای آن‌ها اجتناب می‌کنیم و فقط درباره تغییراتی که در دو تابع gradiant_descent و accuracy توضیح می‌دهیم):

```
def sigmoid(x):
  return 1 / (1+np.exp(-x))

def logistic_regression(x, w):
  y_hat = sigmoid(x @ w)
  return y_hat

def bce(y, y_hat):
  loss = -(np.mean(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)))
  return loss

def gradient(x, y, y_hat):
  grads = (x.T @ (y_hat - y)) / len(y)
  return grads

def  gradiant_descent(w, eta, grads):
  w -= eta*grads.mean(axis=1, keepdims=True)
  return w

def accuracy(y, y_hat):
  acc = np.mean((y == 1) == (np.round(y_hat)))
  return acc
```

در تابع gradiant_descent، میانگین اعداد grads در راستای ستون ۱ (axis=1) گرفته می‌شود و همچنین بخش keepdims=True نیز اطمینان حاصل می‌کند که ابعاد حاصل این عملیات تغییر پیدا نکند که بتوان عملیات آپدیت w را بدون خطا انجام داد.

در تابع accuracy به جای استفاده از روش گفته شده در ویدیو، دقت را به این شکل محاسبه می‌کنیم که چک می‌کنیم اگر مقدار تارگت برابر با یک بود و مقدار رند شده y پیشبینی شده توسط مدل (y_hat) نیز برابر با یک بود، خروجی یک می‌گیریم و درنهایت میانگین می‌گیریم که برابر با دقت مدل می‌باشد.

یک ستون (با اندازه یک) به x_train اضافه می‌کنیم تا در زمان استفاده از آن، به درستی ۵ عدد w حاصل شود:

```
x_train = np.hstack((np.ones((len(x_train), 1)), x_train))
x_train.shape
```

```
(1097, 5)
```

در ادامه برای شروع عملیات، ۵ عدد w (تعداد ویژگی بعلاوه یک) تصادفی تولید می‌کنیم. در این قسمت هم طبق روال قسمت‌های قبل برای تکرارپذیری، random state تعریف می‌کنیم (دو رقم آخر شماره دانشجویی) و همچنین مقدار گام و تعداد تکرار را نیز تعریف می‌کنیم: (این اعداد شاید در ادامه برای گرفتن نتیجه بهتر مدل، دچار تغییر شوند که در صورت نیاز در گزارش ذکر می‌شود.)

```
w = np.random.RandomState(3).randn(5, 1)
print(w)

eta = 0.01
n_epochs = 2000
```

```
[[ 1.78862847]
 [ 0.43650985]
 [ 0.09649747]
 [-1.8634927 ]
 [-0.2773882 ]]
```

مثل ویدیو آموزشی، از توابع تعریف شده در قبل استفاده می‌کنیم و یک تاریخچه هم برای خطا ایجاد می‌کنیم که در ادامه با استفاده از آن بتوانیم نمودار اتلاف را رسم کنیم:

```
error_hist = []

for epoch in range(n_epochs):
  y_hat = logistic_regression(x_train, w)

  e = bce(y_train, y_hat)
  error_hist.append(e)

  grads = gradient(x_train, y_train, y_hat)

  w = gradiant_descent(w, eta, grads)

  if (epoch+1) % 100 == 0:
    print(f'epoch={epoch},  E={e:.3},  w={w.T[0]}')
```

که خروجی آن برای بار اول اجرا (با توجه با آپدیت شدن **w**، با هر بار اجرای این بخش کد، به سابقه اضافه شده و در واقع از مقدار **w** پس از آخرین اجرا استفاده می‌کند و نه مقدار **w** تصادفی تولید شده در ابتدا)، مطابق شکل زیر است:

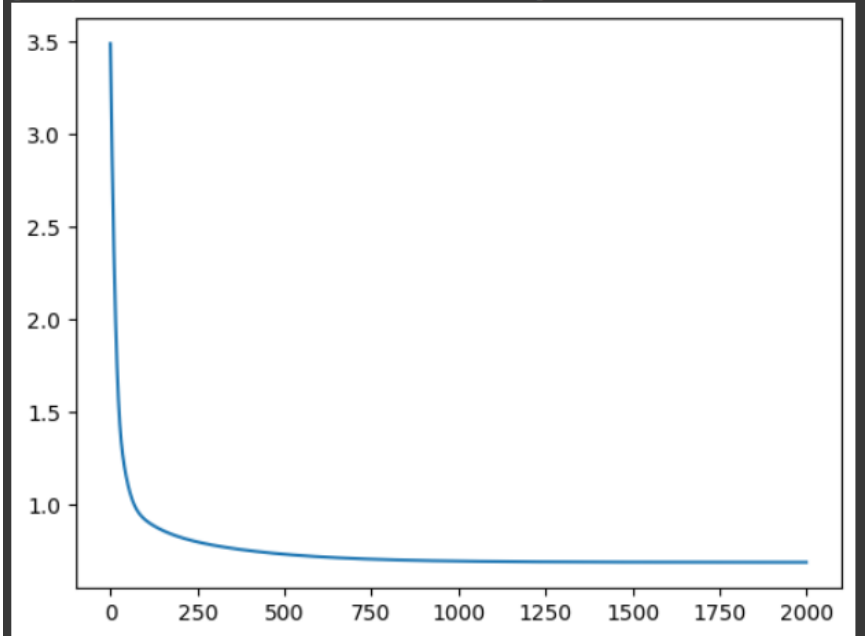
```
epoch=99,  E=0.917,  w=[ 1.65153929 -0.06175842 -0.35989953 -0.4905877  -0.13148056]
epoch=199,  E=0.822,  w=[ 1.35185006 -0.11716424 -0.20558106 -0.3237789   0.05937452]
epoch=299,  E=0.776,  w=[ 1.08348558 -0.12662545 -0.13482281 -0.24621895  0.13655762]
epoch=399,  E=0.748,  w=[ 0.85672114 -0.11717726 -0.09699248 -0.1951565   0.14939267]
epoch=499,  E=0.729,  w=[ 0.66616777 -0.1016028  -0.07449545 -0.15796342  0.13762605]
epoch=599,  E=0.715,  w=[ 0.5066836  -0.08560485 -0.0593759  -0.1291212   0.11885782]
epoch=699,  E=0.706,  w=[ 0.37378445 -0.071214   -0.04820438 -0.1060089   0.09988221]
epoch=799,  E=0.699,  w=[ 0.26346511 -0.05891737 -0.03946886 -0.08720233  0.08294438]
epoch=899,  E=0.694,  w=[ 0.17215416 -0.04863487 -0.03243927 -0.07179747  0.06852995]
epoch=999,  E=0.691,  w=[ 0.09672297 -0.04011505 -0.02670731 -0.05914403  0.05650604]
epoch=1099,  E=0.689,  w=[ 0.03448314 -0.03308213 -0.02200636 -0.04873811  0.0465611 ]
epoch=1199,  E=0.688,  w=[-0.01684    -0.02728455 -0.01814128 -0.04017522  0.03836409]
epoch=1299,  E=0.687,  w=[-0.05915007 -0.02250697 -0.01495976 -0.03312591  0.03161599]
epoch=1399,  E=0.686,  w=[-0.09402865 -0.01856962 -0.01233928 -0.02732046  0.02606194]
epoch=1499,  E=0.686,  w=[-0.12278385 -0.01532395 -0.01018001 -0.02253766  0.02148976]
epoch=1599,  E=0.686,  w=[-0.14649466 -0.01264771 -0.00840019 -0.01859603  0.01772451]
epoch=1699,  E=0.685,  w=[-0.16604987 -0.0104404  -0.00693271 -0.01534658  0.01462256]
epoch=1799,  E=0.685,  w=[-0.18218109 -0.0086194  -0.00572244 -0.01266696  0.01206605]
epoch=1899,  E=0.685,  w=[-0.19549043 -0.00711677 -0.00472405 -0.01045666  0.00995834]
epoch=1999,  E=0.685,  w=[-0.20647351 -0.00587662 -0.00390029 -0.00863307  0.0082201 ]
```

لازم به ذکر است برای بدست آمدن این مقادیر، مقدار گام آموزشی (eta) بر روی ۰.۰۲ تنظیم شده است.

نمودار اتلاف را رسم می‌کنیم:

```
plt.plot(error_hist)
```

```
[<matplotlib.lines.Line2D at 0x7ccb30075420>]
```



با توجه به نمودار اتلاف، مشخص است که سیستم تا حدود ۵۰۰ تکرار اول، مقدار خطا را مقدار چشم‌گیری کم می‌کند و بعد از آن تا تکرار حدود ۱۷۰۰ مقدار کمی خطا را کاهش می‌دهد و در ۳۰۰ تکرار آخر دیگر خطا ثابت شده است. بنظر من نمی‌توان از روی نمودار اتلاف درباره دقت مدل روی داده‌های تست اظهار نظر کرد.

دقت برای داده‌های train:

```
accuracy(y_train, y_hat)
```

```
0.5642661804922516
```

دقت برای داده‌های test: (قبل از محاسبه دقت، یک ستون به x_test اضافه می‌کنیم)

```
x_test = np.hstack((np.ones((len(x_test), 1)), x_test))
x_test.shape
```

```
(275, 5)
```

```
y_hat = logistic_regression(x_test, w)
accuracy(y_test, y_hat)
```

```
0.52
```

بخش ۴.

در بسیاری از الگوریتم‌های مورد استفاده ما، به ویژه الگوریتم‌های مبتنی بر بهینه‌سازی مانند گرادیان نزولی، در یک مقیاس نبودن ویژگی‌ها (Feature) می‌تواند باعث مشکل در فرایند یادگیری شود؛ یعنی ویژگی‌های با اندازه‌های بزرگ‌تر در یادگیری غالب شده و مدل به سمت آنها جهت‌گیری می‌کند پس فرایند نرمال‌سازی باعث از بین رفتن احتمال این خطا شده و باعث می‌شود تاثیر همه‌ی ویژگی‌ها در یادگیری یکسان شود. همچین اگر داده‌ها نرمالیزه شده باشند، امکان تفسیر مدل بر اساس پارامترهای بدست آمده بیشتر (راحت‌تر) می‌شود و در واقع باعث درک نسبی تاثیر هر ویژگی بر مدل می‌شود. همچنین نرمال‌سازی داده‌ها قبل از شروع آموزش مدل، می‌تواند باعث بیشتر شدن سرعت همگرایی نیز شود چون باعث برداشتن گام‌های منسجم‌تری به سمت نقطه پایان می‌شود. از دیگر مزایای نرمالسازی، می‌توان به کم کردن تاثیر داده‌های پرت اشاره کرد.

دو روش نرمالسازی که می‌توان به آن‌ها اشاره کرد، Min-Max Scaling و Z-score Normalization هستند. روش Min-Max Scaling، داده‌ها را در بازه ۰ تا ۱ نرمال می‌کند و از فرمول زیر استفاده می‌کند:

$$x = \frac{x - x_{min}}{x_{max} - x_{min}}$$

و روش Z-score Normalization به شکل زیر نرمالسازی را انجام می‌دهد:

$$x = \frac{x - \mu}{\sigma}$$

که در این فرمول، μ میانگین و σ واریانس می‌باشند که در نهایت داده‌های نرمال شده دارای میانگین ۰ و واریانس ۱ خواهند بود.

در ادامه برای نرمالسازی از روش اول استفاده خواهد شد که در سایت scikit-learn دارای کتابخانه آماده sklearn.preprocessing.minmax_scale است و به شکل زیر مورد استفاده قرار می‌گیرد و در ادامه داده‌های نرمالایز شده نیز نمایش داده شده‌اند:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
x_tarinScaled = scaler.fit_transform(x_train)
x_testScaled = scaler.transform(x_test)
print(x_tarinScaled)
print(x_testScaled)
```

```
[[0.         0.18491516 0.80097205 0.21426842 0.16119734]
 [0.         0.79118621 0.41130321 0.32304478 0.80529292]
 [0.         0.79220302 0.61652107 0.29326039 0.85937969]
 ...
 [0.         0.56124801 0.58728379 0.0900252  0.76402339]
 [0.         0.80786621 0.94677208 0.07204859 0.40611219]
 [0.         0.7346487  0.74241414 0.20555754 0.64138865]]
[[0.         0.71011545 0.54362182 0.28554505 0.88759468]
 [0.         0.24870735 0.53045254 0.25841428 0.74697437]
 [0.         0.51110227 0.7768261  0.27384496 0.69241751]
 ...
 [0.         0.88584327 0.92098205 0.0503457  0.38812661]
 [0.         0.79921251 0.54127011 0.2719969  0.87783809]
 [0.         0.59300925 0.83938055 0.13911732 0.69512262]]
```

بخش ۵.

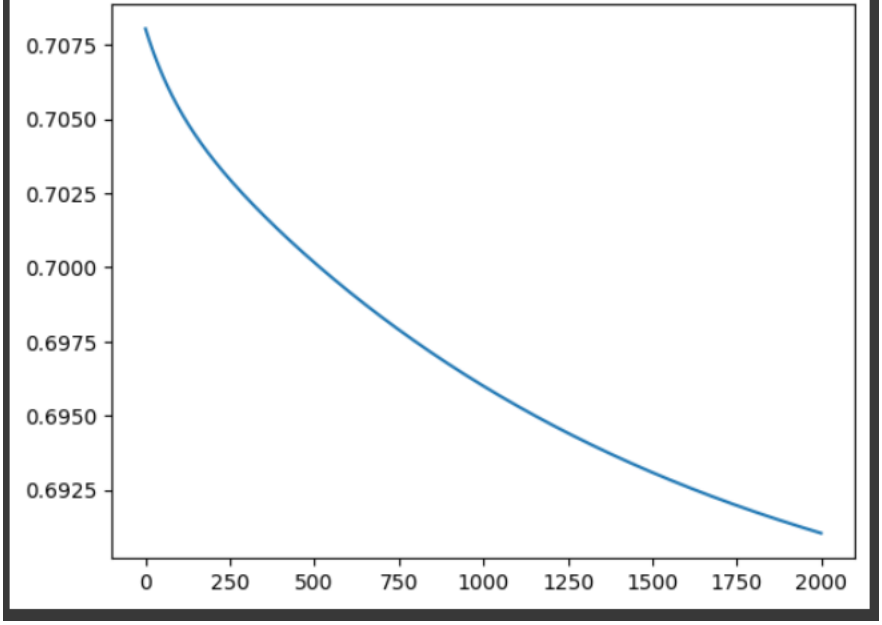
چون منطق کدزنی برای تکرار مراحل دچار تغییری نمی‌شود، از آوردن تصویر کدها خودداری شده و فقط نتایج در گزارشکار آورده شده‌اند. کدها در فایل گوگل کولب قابل مشاهده هستند. همچنین لازم به ذکر است که در مراحل قبل فرایند بر زدن و تقسیم داده‌ها به train و test و اضافه کردن یک ستون یک انجام شده است و ما نرمالایز کردن را روی این داده‌ها انجام دادیم پس نیازی به تکرار این قسمت‌ها نداریم.

نتایج مدل:

```
epoch=99,  E=0.705,  w=[ 1.78862847  0.44925102  0.09866609 -1.8140666  -0.2250453 ]
epoch=199,  E=0.704,  w=[ 1.78862847  0.45070956  0.08950661 -1.77098537 -0.18736456]
epoch=299,  E=0.702,  w=[ 1.78862847  0.44592325  0.07440336 -1.73194502 -0.15835453]
epoch=399,  E=0.701,  w=[ 1.78862847  0.43775695  0.05640026 -1.69561161 -0.13458229]
epoch=499,  E=0.7,  w=[ 1.78862847  0.42782574  0.03719887 -1.66121649 -0.11409347]
epoch=599,  E=0.699,  w=[ 1.78862847  0.41703496  0.01773994 -1.62831471 -0.09577593]
epoch=699,  E=0.698,  w=[ 1.78862847e+00  4.05889592e-01 -1.46419104e-03 -1.59664499e+00
 -7.89936449e-02]
epoch=799,  E=0.697,  w=[ 1.78862847  0.39466963 -0.02014131 -1.56604975 -0.06337876]
epoch=899,  E=0.697,  w=[ 1.78862847  0.38352897 -0.0381534  -1.5364298  -0.0487141 ]
epoch=999,  E=0.696,  w=[ 1.78862847  0.37255099 -0.05543706 -1.50771887 -0.03486714]
epoch=1099,  E=0.695,  w=[ 1.78862847  0.36177965 -0.07197024 -1.47986927 -0.02175299]
epoch=1199,  E=0.695,  w=[ 1.78862847  0.35123691 -0.08775357 -1.45284383 -0.00931368]
epoch=1299,  E=0.694,  w=[ 1.78862847  0.34093248 -0.10279998 -1.4266115   0.00249346]
epoch=1399,  E=0.694,  w=[ 1.78862847  0.33086929 -0.11712888 -1.4011448   0.01370223]
epoch=1499,  E=0.693,  w=[ 1.78862847  0.32104644 -0.13076294 -1.37641848  0.02434116]
epoch=1599,  E=0.693,  w=[ 1.78862847  0.31146099 -0.14372628 -1.35240872  0.03443556]
epoch=1699,  E=0.692,  w=[ 1.78862847  0.30210884 -0.1560435  -1.32909272  0.04400862]
epoch=1799,  E=0.692,  w=[ 1.78862847  0.29298529 -0.16773908 -1.30644851  0.05308203]
epoch=1899,  E=0.691,  w=[ 1.78862847  0.28408533 -0.17883715 -1.28445476  0.06167636]
epoch=1999,  E=0.691,  w=[ 1.78862847  0.27540381 -0.18936124 -1.26309078  0.06981126]
```

نمودار اتلاف:

```
[<matplotlib.lines.Line2D at 0x7d0832b2eb90>]
```



دقت‌ها:

```
accuracy(y_train, y_hat)
```

```
0.5525814204027513
```

```
y_hat = logistic_regression(x_testScaled, w)
accuracy(y_test, y_hat)
```

```
0.5163636363636364
```

بخش ۶.

برای چک کردن وضعیت تعادل داده‌ها، تعداد داده‌های موجود در هر کلاس را شمارش می‌کنیم و سپس آن‌ها را مقایسه می‌کنیم:

```
class_counts = df['Class'].value_counts()
print("Class Distribution:")
print(class_counts)
```

```
Class Distribution:
0    762
1    610
Name: Class, dtype: int64
```

همانطور که مشخص است، داده‌ها متعادل نیستند و داده‌های کلاس صفر دارای تعداد بیشتری می‌باشند.

عدم تعادل در داده‌ها می‌تواند موجب مشکلاتی شود مانند اینکه مدل برای داده‌های کلاس اکثریت بهتر عمل می‌کند و داده‌های کلاس اقلیت را در نظر نمی‌گیرد و برای آن‌ها ضعیف‌تر عمل می‌کند؛ همچنین داده‌های نامتعادل می‌تواند باعث ایجاد مدلی شود که برای الگوهای خاص به درستی کار کند، یعنی جامعیت مدل از بین می‌رود و در مواجهه با دیتاست‌های دیگر به خوبی عمل نکند. همچنین دقت دیگر معیار خوبی برای ارزیابی کیفیت مدل نیست زیرا شاید برای داده‌های کلاس اکثریت به خوبی عمل کند ولی برای کلاس اقلیت احتمالا نتیجه مناسبی نخواهد داشت.

برای حل این مشکل، می‌توان از راه‌های مختلفی استفاده کرد؛ مثلا برای کلاس با تعداد داده کمتر، تعدادی داده تولید کرد (به این شکل که با میانگین داده‌های کلاس اقلیت و واریانس کم تعدادی داده تولید کرد)، یا اگر اختلاف تعداد دو کلاس خیلی زیاد نبود، تعدادی از داده‌های کلاس اکثریت را حذف نمود و کارهایی از این قبیل. در ادامه برای رفع مشکل، با توجه به اینکه اختلاف تعداد دو کلاس ۱۵۲ عدد است و کلاس اقلیت ۶۱۰ داده دارد، می‌توان گفت روش بهتر، حذف ۱۵۲ داده از کلاس اکثریت است تا هر دو کلاس ۶۱۰ داده داشته باشند.

```
target_count = int(min(class_counts))

excess_samples = class_counts[0] - target_count
if excess_samples > 0:
    indices_to_delete = np.random.choice(df_shuffled[df_shuffled['Class'] == 0].index, size=excess_samples, replace=False)

    df_balanced = df_shuffled.drop(indices_to_delete)

new_class_counts = df_balanced['Class'].value_counts()
print("New Class Distribution:")
print(new_class_counts)
```

در این قسمت کد، در ابتدا تعداد داده‌های اقلیت را به عنوان هدف (target_count) تعریف می‌کنیم؛ سپس تعداد داده‌هایی را که باید از کلاس اکثریت حذف کنیم را مشخص می‌کنیم. (تعداد داده‌های کلاس صفر منهای تعداد داده های کلاس یک یا همان هدف) در ادامه به تعداد این اختلاف (۱۵۲) داده به شکل تصادفی از داده‌های کلاس اکثریت انتخاب می‌کنیم و پس از آن داده‌های انتخاب شده را از دیتاست حذف می‌کنیم. در انتها برای اطمینان از درست انجام شدن مراحل، تعداد داده‌ها را دوباره می‌شماریم:

```
New Class Distribution:
1    610
0    610
Name: Class, dtype: int64
```

بخش ۷.

با توجه به تکراری بودن روند کدزنی، توضیحات خلاصه گفته می‌شود.

در ابتدا از دیتافریم جدید، ویژگی‌ها و لیبل‌ها را جدا می‌کنیم و سپس آن‌ها را به دو بخش train و test تقسیم می‌کنیم:

```
X_balanced = df_balanced[['Feature1', 'Feature2', 'Feature3', 'Feature4']].values
y_balanced = df_balanced['Class'].values
X_balanced, y_balanced
```

```
(array([[-0.62043,  0.5587 , -0.38587, -0.66423],
        [ 4.7965 ,  6.9859 , -1.9967 , -0.35001],
        [ 4.0422 , -4.391  ,  4.7466 ,  1.137  ],
        ...,
        [-1.9555 ,  0.20692,  1.2473 , -0.3707 ],
        [-1.2846 ,  3.2715 , -1.7671 , -3.2608 ],
        [-2.3675 , -0.43663,  1.692  , -0.43018]]),
 array([1, 0, 0, ..., 1, 1, 1]))
```

```
from sklearn.model_selection import train_test_split

x_trainBalanced, x_testBalanced, y_trainBalanced, y_testBalanced = train_test_split(X_balanced, y_balanced, test_size=0.2, random_state=3)
x_trainBalanced.shape, x_testBalanced.shape, y_trainBalanced.shape, y_testBalanced.shape
```

```
((976, 4), (244, 4), (976,), (244,))
```

سپس مدل را با LogisticRegression، آموزش می‌دهیم:

```
model_Q2 = LogisticRegression(solver='saga', max_iter=2000, random_state=3)
model_Q2.fit(x_trainBalanced, y_trainBalanced)
model_Q2.predict(x_testBalanced), y_testBalanced
```

در ادامه دقت مدل برای داده‌های train و test را ملاحظه می‌کنیم:

```
model_Q2.score(x_trainBalanced, y_trainBalanced)
```
```
0.9897540983606558
```
```
model_Q2.score(x_testBalanced, y_testBalanced)
```
```
0.9959016393442623
```

سوال سوم)

بخش ۱.

این دیتاست، شامل مجموعه‌ای از شاخصه‌های آماری مرتبط با سلامت مانند داشتن فشار خون بالا، داشتن کلسترول بالا، شاخص توده بدنی و ... و عوامل سبک زندگی (۲۱ شاخصه) می‌باشد که بعضی از آنها به صورت باینری و برخی دیگر، غیر باینری هستند. هدف از استفاده از این دیتاست، پیش‌بینی احتمال دچار شدن فرد به حمله قلبی یا بیماری قلبی می‌باشد که در ستون اول دیتاست آورده شده است و همان لیبل ما حساب می‌شود.

بارگذاری در محیط گوگل کولب:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1PLBxkbNgW7v3kuPB32PPVc7L4h3WQHfW
```

فراخوانی کتابخانه‌های مورد نیاز و مشاهده دیتافریم اولیه دیتاست قبل از استخراج ۱۰۰ نمونه مربوط به هر کلاس:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
df = pd.read_csv('/content/heart_disease_health_indicators.csv')
df
```

	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...
0	0	1	1	1	40	1	0	0	0	0	...
1	0	0	0	0	25	1	0	0	1	0	...
2	0	1	1	1	28	0	0	0	0	1	...
3	0	1	0	1	27	0	0	0	1	1	...
4	0	1	1	1	24	0	0	0	1	1	...
...	...	...	...	...	...	...	...	...	...	...	...
253656	0	0	0	1	25	0	0	0	1	1	...
253657	0	0	1	1	24	0	0	0	0	0	...
253658	0	0	0	0	27	0	0	0	1	0	...
253659	0	0	1	1	37	0	0	2	0	0	...
253660	0	0	1	1	34	1	0	0	0	1	...

253661 rows × 22 columns

بخش ۲.

در این بخش از دیتاست اصلی، ۱۰۰ نمونه از کلاس یک و ۱۰۰ نمونه از کلاس صفر جدا می‌کنیم. (ستون کلاس در واقع همان ستون HeartDiseaseorAttack می‌باشد) برای این کار ابتدا با متود sample. که به صورت تصادفی نمونه انتخاب می‌کند، یک دیتافریم از صد نمونه از داده‌های کلاس یک و یک دیتافریم از صد نمونه از داده‌های کلاس صفر تشکیل می‌دهیم. (برای تکرارپذیر شدن نتایج، random state را برابر با دو رقم آخر شماره دانشجویی قرار می‌دهیم):

```
class_1_df = df[df['HeartDiseaseorAttack'] == 1].sample(n=100, random_state=3)
class_0_df = df[df['HeartDiseaseorAttack'] == 0].sample(n=100, random_state=3)
```

در ادامه با متود concat. دو دیتافریم تشکیل شده در بالا را، به هم می‌چسبانیم ولی مشکل این کار این است که صد نمونه مربوط به کلاس یک پشت هم و صد نمونه مربوط به کلاس صفر در ادامه آن آورده می‌شود. برای جلوگیری از این اتفاق، با استفاده از دستور sample(frac=1)، داده‌ها را بر می‌زنیم:

```
new_df = pd.concat([class_1_df, class_0_df])

new_df = new_df.sample(frac=1, random_state=42).reset_index(drop=True)

print(new_df)
```

دیتافریم نهایی که در بخش‌های بعدی مورد استفاده قرار می‌گیرد، به صورت زیر خواهد بود:

	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	\
0	1	1	1	1	36	0	0	
1	1	1	1	1	37	1	0	
2	1	1	1	1	43	1	0	
3	0	1	0	1	25	0	0	
4	0	1	1	1	28	0	0	
..	...	...	...	...	...	...	...	
195	0	0	0	1	26	1	0	
196	1	1	1	1	36	0	0	
197	1	1	1	1	28	0	0	
198	0	1	1	1	31	1	0	
199	0	0	1	1	31	1	0	

بقیه‌ی ویژگی‌ها هم در محیط کولب قابل مشاهده است که برای جلوگیری از شلوغ شدن گزارش کار در اینجا آورده نشده.

بخش ۳.

برای استفاده از طبقه‌بندی‌های آماده، ابتدا ویژگی‌ها را در پارامتر X و تارگت را در y ذخیره می‌کنیم:

```
X = new_df[['HighBP', 'HighChol', 'CholCheck', 'BMI',
                'Smoker', 'Stroke', 'Diabetes', 'PhysActivity',
                'Fruits', 'Veggies', 'HvyAlcoholConsump', 'AnyHealthcare',
                'NoDocbcCost', 'GenHlth', 'MentHlth', 'PhysHlth',
                'DiffWalk', 'Sex', 'Age', 'Education',
                'Income']].values
y = new_df['HeartDiseaseorAttack'].values
X, y
```

```
(array([[ 1,  1,  1, ...,  9,  6,  6],
        [ 1,  1,  1, ..., 11,  4,  3],
        [ 1,  1,  1, ...,  5,  5,  5],
        ...,
        [ 1,  1,  1, ...,  3,  4,  8],
        [ 1,  1,  1, ...,  4,  6,  1],
        [ 0,  1,  1, ...,  5,  6,  8]]),
 array([1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1,
        0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1,
        0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0,
        1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1,
        0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
        1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,
        0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        0, 0]))
```

برای طبقه بندی طبق روال سوالات قبل از LogisticRegression و SGDClassifier استفاده می‌کنیم.
ابتدا داده‌ها را به دو بخش train و test تقسیم می‌کنیم:

```
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.model_selection import train_test_split
```

LogisticRegression:

```
model1 = LogisticRegression(solver='saga', max_iter=2500, random_state=3)
model1.fit(x_train, y_train)
model1.predict(x_test), y_test
```

```
(array([1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
        0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0]),
 array([1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
        0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1]))
```

که دقت‌های زیر را نتیجه می‌دهد:

```
model1.score(x_train, y_train)
```

```
0.7875
```

```
model1.score(x_test, y_test)
```

```
0.725
```

SGDClassifier:

```
model2 = SGDClassifier(loss='log_loss', max_iter=500, learning_rate='constant', eta0=0.2, random_state=3)
model2.fit(x_train, y_train)
model2.predict(x_test), y_test
```

```
(array([1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
        0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0]),
 array([1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
        0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1]))
```

دقت:

```
model2.score(x_train, y_train)
```

```
0.7125
```

```
model2.score(x_test, y_test)
```

```
0.65
```